

sshmount ソースリスト

美都

2026 年 1 月 12 日

目次

1	メインモジュール main.rs	2
2	コマンドラインオプションの定義 cmdline_opt.rs	3
3	コマンドライン remotehost の解析部 cmdline_opt/remote_host.rs	5
4	ssh2 ログイン処理モジュール ssh_connect.rs	13
5	FUSE 接続オプション生成モジュール fuse_util.rs	18
6	ファイルシステムモジュール ssh_filesystem.rs	21
7	ファイルハンドル管理モジュール ssh_filesystem/file_handle.rs	38
8	Inode 管理モジュール ssh_filesystem/inode.rs	39
9	双方向ハッシュマップモジュール ssh_filesystem/bi_hash_map.rs	43

1 メインモジュール main.rs

```
1  mod cmdline_opt;
2  mod fuse_util;
3  mod ssh_connect;
4  mod ssh_filesystem;
5
6  use anyhow::{Context, Result};
7  use clap::Parser;
8  use cmdline_opt::Opt;
9  use daemonize::Daemonize;
10 use fuse_util::{make_full_path, make_mount_option, make_remote_path};
11 use ssh_connect::make_ssh_session;
12 //use log::debug;
13
14 fn main() -> Result<()> {
15     env_logger::init();
16     let opt = Opt::parse();
17
18     let ssh = make_ssh_session(&opt).context("Failed to generate ssh session.")?;
19
20     let path = make_remote_path(&opt, &ssh).context("Failed to generate remote path.")?;
21     let options = make_mount_option(&opt);
22     let mount_point = make_full_path(&opt.mount_point)?;
23
24     // プロセスのデーモン化
25     if opt.daemon {
26         let daemonize = Daemonize::new();
27         if let Err(e) = daemonize.start() {
28             eprintln!("daemonization failed.(error: {})", e);
29         }
30     }
31     // ファイルシステムへのマウント実行
32     let fs = ssh_filesystem::Sshfs::new(ssh, &path)?;
33     fuser::mount2(fs, mount_point, &options).context("Failed to mount FUSE.")?;
34     Ok(())
35 }
```

2 コマンドラインオプションの定義 cmdline_opt.rs

```
1  pub mod remote_host;
2
3  use anyhow::{anyhow, Context};
4  use clap::Parser;
5  use std::path::PathBuf;
6
7  use remote_host::RemoteName;
8
9  /// コマンドラインオプション
10 #[derive(Parser)]
11 #[command(author, version, about)]
12 pub struct Opt {
13     /// Distination [user@]host:[path] or scp://[user@]host[:port][/path]
14     pub remote: RemoteName,
15     /// Path to mount
16     #[arg(value_parser = exist_dir)]
17     pub mount_point: String,
18     /// Path to config file
19     #[arg(short = 'F', long)]
20     pub config_file: Option<PathBuf>,
21     /// Login name
22     #[arg(short, long)]
23     pub login_name: Option<String>,
24     /// File name of secret key file
25     #[arg(short, long)]
26     pub identity: Option<PathBuf>,
27     /// Port no
28     #[arg(short, long)]
29     pub port: Option<u16>,
30     /// Read only
31     #[arg(short, long)]
32     pub readonly: bool,
33     /// Not executable
34     #[arg(long)]
35     pub no_exec: bool,
36     /// Do not change access date and time(ctime)
37     #[arg(long)]
38     pub no_atime: bool,
39     /// run in daemon mode
40     #[arg(short, long)]
41     pub daemon: bool,
42 }
43
```

```

44  /// 指定されたディレクトリが存在し、中にファイルがないことを確認する。
45  fn exist_dir(s: &str) -> anyhow::Result<String> {
46      match std::fs::read_dir(s) {
47          Ok(mut dir) => match dir.next() {
48              None => Ok(s.to_string()),
49              Some(_) => Err(anyhow!("Mount destination directory is not empty.")),
50          },
51          Err(e) => match e.kind() {
52              std::io::ErrorKind::NotFound => Err(anyhow!("The mount directory does not exist.")),
53              std::io::ErrorKind::NotConnected => Err(anyhow!(
54                  "The network of the mount directory is disconnected. (Did you forget to umount?)."
55              )),
56              _ => Err(e).context("Unexpected error.(check mount directory)"),
57          },
58      }
59  }
60
61  #[cfg(test)]
62  mod test {
63      use super::*;
64      #[test]
65      fn verify_cli() {
66          use clap::CommandFactory;
67          Opt::command().debug_assert()
68      }
69  }

```

3 コマンドライン remotehost の解析部 cmdline_opt/remote_host.rs

```
1  /// コマンドラインオプションにおける RemoteHost 構造体
2
3  use fluent_uri::{component::Host, Uri};
4  use log::debug;
5  use std::net::IpAddr;
6
7  /// コマンドラインの接続先ホスト情報
8  #[derive(Clone, Debug, PartialEq)]
9  pub struct RemoteName {
10     /// ユーザー名
11     pub user: Option<String>,
12     /// IP アドレス
13     pub host: HostInfo,
14     /// ポート番号
15     pub port: Option<u16>,
16     /// 接続先パス
17     pub path: Option<std::path::PathBuf>,
18 }
19
20 /// ホスト情報 (ホスト名または、IP アドレス)
21 #[derive(Clone, Debug, PartialEq)]
22 pub enum HostInfo {
23     Name(String),
24     Ip(IpAddr),
25 }
26
27 impl RemoteName {
28     /// remote 引数の解析 (URI 形式の場合)
29     fn parse_uri(s: &str) -> Result<Self, ErrorRemoteName> {
30         // URI の解析
31         let uri = Uri::parse(s)?;
32         let info = uri
33             .authority()
34             .ok_or(ErrorRemoteName::NotFoundHostInformationInURI)?;
35         // ポート番号の取得
36         let port = info
37             .port_to_u16()
38             .map_err(|_| ErrorRemoteName::InvalidPortNo)?;
39         // ホストアドレスの取得
40         let host = match info.host_parsed() {
41             Host::Ipv4(ip) => HostInfo::Ip(ip.into()),
42             Host::Ipv6(ip) => HostInfo::Ip(ip.into()),
43             _ => HostInfo::Name(info.host().to_string()),
```

```

44     };
45     // パス名の取得
46     let path_str = uri.path().as_str();
47     let path = if path_str.is_empty() {
48         None
49     } else {
50         Some(
51             path_str
52                 .trim()
53                 .parse::<std::path::PathBuf>()
54                 .map_err(|_| ErrorRemoteName::InvalidPath)?,
55         )
56     };
57     // ユーザー名の取得
58     let user = info.userinfo().map(|s| s.as_str().trim().to_string());
59
60     Ok(Self {
61         user,
62         host,
63         port,
64         path,
65     })
66 }
67
68 /// remote 引数の解析 (URI 形式でない場合)
69 fn parse_non_uri(s: &str) -> Result<Self, ErrorRemoteName> {
70     let mut rest_str = s.trim();
71     // ユーザー名の取得
72     let user = match rest_str.split_once("@") {
73         Some((u, r)) => {
74             rest_str = r;
75             Some(u.trim().to_string())
76         }
77         None => None,
78     };
79     // ホストアドレスの取得
80     let host_str: &str;
81     // IPv6 の判定
82     rest_str = rest_str.trim_start();
83     if rest_str.starts_with('[') {
84         let (ip6_str, rest) = match rest_str.split_once(']') {
85             Some((l, r)) => (l.trim_start_matches('['), r),
86             None => return Err(ErrorRemoteName::MissingClosingBracketInIPv6),
87         };
88         host_str = ip6_str.trim();
89         let (_, r) = rest.split_once(":").ok_or(ErrorRemoteName::NoColon)?;

```

```

90         rest_str = r;
91     } else {
92         let (h, r) = rest_str.split_once(':',').ok_or(ErrorRemoteName::NoColon)?;
93         if h.trim().is_empty() {
94             return Err(ErrorRemoteName::NoHostName);
95         }
96         host_str = h.trim();
97         rest_str = r;
98     }
99
100     let try_ip = host_str.parse::<IpAddr>();
101     let host = match try_ip {
102         Ok(addr) => HostInfo::Ip(addr),
103         Err(_) => HostInfo::Name(host_str.to_string()),
104     };
105
106     // パス名の取得
107     rest_str = rest_str.trim();
108     let path = if rest_str.is_empty() {
109         None
110     } else {
111         Some(
112             rest_str
113                 .parse::<std::path::PathBuf>()
114                 .map_err(|_| ErrorRemoteName::InvalidPath)?,
115         )
116     };
117
118     Ok(Self {
119         host,
120         port: None,
121         user,
122         path,
123     })
124 }
125
126
127 impl std::fmt::Display for RemoteName {
128     fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
129         let s = format!(
130             "<{:?}><{:?}><{:?}><{:?}>",
131             &self.user, &self.host, &self.port, &self.path
132         );
133         s.fmt(f)
134     }
135 }

```

```

136
137 impl std::str::FromStr for RemoteName {
138     type Err = ErrorRemoteName;
139     fn from_str(s: &str) -> Result<Self, Self::Err> {
140         let ret = if s.trim_start().starts_with("scp://") {
141             Self::parse_uri(s)
142         } else {
143             Self::parse_non_uri(s)
144         };
145         debug!("RemoteName: {:?}", ret);
146         ret
147     }
148 }
149
150 #[derive(thiserror::Error, Debug, PartialEq, Eq)]
151 //#[error("The format of the host to connect to is \"[user@]host:[path]\" or
152 ↪ \"scp://[user@]host[:port][/path]\".")
153 pub enum ErrorRemoteName {
154     #[error("URI parse error.")]
155     UriParse(#[from] fluent_uri::ParseError),
156     #[error("Host information not found at the specified URI.")]
157     NotFoundHostInformationInURI,
158     #[error("Invalid port number.")]
159     InvalidPortNo,
160     #[error("Invalid path name")]
161     InvalidPath,
162     #[error("The closing bracket is missing from IPv6.")]
163     MissingClosingBracketInIPv6,
164     #[error("In the host information, there is no subsequent\":\\\"")]
165     NoColon,
166     #[error("No hostname specified.")]
167     NoHostName,
168 }
169
170 #[cfg(test)]
171 mod test {
172     use fluent_uri::ParseErrorKind;
173
174     use super::*;
175
176     #[test]
177     fn test_from_str_remotename() {
178         use std::path::Path;
179         let s = "mito@reterminal.local:/home/mito";
180         let r: RemoteName = s.parse().unwrap();
181         let k = RemoteName {

```

```

181         user: Some("mito".to_string()),
182         host: HostInfo::Name("reterminal.local".to_string()),
183         port: None,
184         path: Some(Path::new("/home/mito").into()),
185     };
186     assert_eq!(r, k);
187
188     let s = "mito@reterminal.local:/home/mito/";
189     let r: RemoteName = s.parse().unwrap();
190     let k = RemoteName {
191         user: Some("mito".to_string()),
192         host: HostInfo::Name("reterminal.local".to_string()),
193         port: None,
194         path: Some(Path::new("/home/mito").into()),
195     };
196     assert_eq!(r, k);
197
198     let s = "mito@[fe80::a00:27ff:fe0e:8c0c]:/home/mito/";
199     let r: RemoteName = s.parse().unwrap();
200     let k = RemoteName {
201         user: Some("mito".to_string()),
202         host: HostInfo::Ip("fe80::a00:27ff:fe0e:8c0c".parse().unwrap()),
203         port: None,
204         path: Some(Path::new("/home/mito").into()),
205     };
206     assert_eq!(r, k);
207
208     let s = "mito@[::1]:/home/mito/";
209     let r: RemoteName = s.parse().unwrap();
210     let k = RemoteName {
211         user: Some("mito".to_string()),
212         host: HostInfo::Ip("::1".parse().unwrap()),
213         port: None,
214         path: Some(Path::new("/home/mito").into()),
215     };
216     assert_eq!(r, k);
217
218     let s = "reterminal.local:";
219     let r: RemoteName = s.parse().unwrap();
220     let k = RemoteName {
221         user: None,
222         host: HostInfo::Name("reterminal.local".to_string()),
223         port: None,
224         path: None,
225     };
226     assert_eq!(r, k);

```

```

227
228     let s = " mito @reterminal.local: ";
229     let r: RemoteName = s.parse().unwrap();
230     let k = RemoteName {
231         user: Some("mito".to_string()),
232         host: HostInfo::Name("reterminal.local".to_string()),
233         port: None,
234         path: None,
235     };
236     assert_eq!(r, k);
237
238     let s = "reterminal.local";
239     let r: Result<RemoteName, ErrorRemoteName> = s.parse();
240     assert_eq!(r, Err(ErrorRemoteName::NoColon));
241
242     let s = "mito@reterminal.local";
243     let r: Result<RemoteName, ErrorRemoteName> = s.parse();
244     assert_eq!(r, Err(ErrorRemoteName::NoColon));
245
246     let s = " mito @: ";
247     let r: Result<RemoteName, ErrorRemoteName> = s.parse();
248     assert_eq!(r, Err(ErrorRemoteName::NoHostName));
249 }
250
251 #[test]
252 // scp://[user@]host[:port][/path] の解析テスト
253 fn test_from_str_remotename_uri() {
254     let s = "scp://name@hostname.hoge:22/test_path";
255     let a = RemoteName {
256         user: Some("name".to_string()),
257         host: HostInfo::Name("hostname.hoge".to_string()),
258         port: Some(22),
259         path: Some(std::path::PathBuf::from("/test_path")),
260     };
261     let r: RemoteName = s.parse().unwrap();
262     assert_eq!(r, a);
263
264     let s = "scp://name@192.168.0.1:22/test_path/path";
265     let a = RemoteName {
266         user: Some("name".to_string()),
267         host: HostInfo::Ip("192.168.0.1".parse().unwrap()),
268         port: Some(22),
269         path: Some(std::path::PathBuf::from("/test_path/path")),
270     };
271     let r: RemoteName = s.parse().unwrap();
272     assert_eq!(r, a);

```

```

273
274     let s = "scp://[::1]:22/test";
275     let a = RemoteName {
276         user: None,
277         host: HostInfo::Ip("::1".parse().unwrap()),
278         port: Some(22),
279         path: Some(std::path::PathBuf::from("/test")),
280     };
281     let r: RemoteName = s.parse().unwrap();
282     assert_eq!(r, a);
283
284     let s = "scp://localhost";
285     let a = RemoteName {
286         user: None,
287         host: HostInfo::Name("localhost".to_string()),
288         port: None,
289         path: None,
290     };
291     let r: RemoteName = s.parse().unwrap();
292     assert_eq!(r, a);
293 }
294
295 #[test]
296 fn test_from_str_with_extra_space() {
297     let s = "name@192.168.0.1: /test_path/path ";
298     let a = RemoteName {
299         user: Some("name".to_string()),
300         host: HostInfo::Ip("192.168.0.1".parse().unwrap()),
301         port: None,
302         path: Some(std::path::PathBuf::from("/test_path/path")),
303     };
304     let r: RemoteName = s.parse().unwrap();
305     assert_eq!(r, a);
306
307     let s = "scp://[::1]:22/ /test";
308     let r: Result<RemoteName, ErrorRemoteName> = s.parse();
309     match r {
310         Ok(_) => {
311             unreachable!("この形式はエラーであるはず。");
312         }
313         Err(ErrorRemoteName::UriParse(e)) => {
314             assert_eq!(e.kind(), ParseErrorKind::UnexpectedChar);
315         }
316         Err(e) => {
317             unreachable!("何が起こった?({:?})", e);
318         }
319     }

```

```

319     }
320
321     let s = "scp:// name @192.168.0.1:22/test_path/path";
322     let r: Result<RemoteName, ErrorRemoteName> = s.parse();
323     match r {
324         Ok(_) => {
325             unreachable!("この形式はエラーであるはず。");
326         }
327         Err(ErrorRemoteName::UriParse(e)) => {
328             assert_eq!(e.kind(), ParseErrorKind::UnexpectedChar);
329         }
330         Err(e) => {
331             unreachable!("何が起こった?({:?})", e);
332         }
333     }
334 }
335 }

```

4 ssh2 ログイン処理モジュール ssh_connect.rs

```
1  /// ssh 接続関連関数モジュール
2
3  use crate::cmdline_opt::{remote_host::HostInfo, Opt};
4  use anyhow::{anyhow, Context, Result};
5  use dialoguer::Password;
6  use dns_lookup::lookup_host;
7  use log::{debug, error};
8  use ssh2::Session;
9  use ssh2_config::{DefaultAlgorithms, HostParams, ParseRule, SshConfig};
10 use std::{
11     fs::File,
12     io::BufReader,
13     net::TcpStream,
14     path::{Path, PathBuf},
15     str,
16 };
17
18 /// セッションを生成する。
19 pub fn make_ssh_session(opt: &Opt) -> Result<Session> {
20     let (address, host_params) = match opt.remote.host {
21         HostInfo::Name(ref name) => {
22             let host_params = get_ssh_config(&opt.config_file).query(name);
23             let address =
24                 get_address(name, opt, &host_params).context("Failed to get host address")?;
25             (address, host_params)
26         }
27         HostInfo::Ip(ip) => {
28             let address = std::net::SocketAddr::from((
29                 ip,
30                 opt.port.unwrap_or(opt.remote.port.unwrap_or(DEFAULT_PORT)),
31             ));
32             let host_params = HostParams::new(&DefaultAlgorithms::default());
33             (address, host_params)
34         }
35     };
36     let username = get_username(opt, &host_params).context("Failed to get user name.")?;
37     debug!(
38         "[main] 接続先情報-> ユーザー:\"{}\", ip address:{:?}\",
39         &username, &address
40     );
41     let identity_file = get_identity_file(opt, &host_params)?;
42
43     let ssh = connect_ssh(address).context("The ssh connection failed.")?;
```

```

44     userauth(&ssh, &username, &identity_file).context("User authentication failed.")?;
45     Ok(ssh)
46 }
47
48 /// ホストの ip アドレス解決
49 const DEFAULT_PORT: u16 = 22;
50 fn get_address(name: &str, opt: &Opt, host_params: &HostParams) -> Result<std::net::SocketAddr> {
51     let dns = host_params.host_name.as_deref().unwrap_or(name);
52     let addr = lookup_host(dns)
53         .inspect_err(|e| error!("get_address : Failed lookup_host[{}]", e))
54         .context("Cannot find host to connect to.")?
55         .collect:::<Vec<_>>();
56     let addr = addr
57         .first()
58         .ok_or(anyhow!("Unable to obtain DNS address. "))
59         .inspect_err(|e| error!("get_address : {}", e))?;
60     Ok(std::net::SocketAddr::from((
61         *addr,
62         opt.port.unwrap_or(opt.remote.port.unwrap_or(DEFAULT_PORT)),
63     )))
64 }
65
66 /// ssh-config の取得と解析
67 /// ファイル名が指定されていない場合は "~/.ssh/config" を使用
68 /// config ファイルのエラー及びファイルがない場合、デフォルト値を返す。
69 fn get_ssh_config(file_opt: &Option<PathBuf>) -> SshConfig {
70     get_config_file(file_opt)
71         .map(BufReader::new)
72         .map_or(SshConfig::default(), |mut f| {
73             SshConfig::default()
74                 .parse(&mut f, ParseRule::ALLOW_UNKNOWN_FIELDS)
75                 .unwrap_or_else(|e| {
76                     eprintln!("警告:config ファイル内にエラー -- {e}");
77                     SshConfig::default()
78                 })
79         })
80 }
81
82 /// ssh_config ファイルがあれば、オープンする。
83 /// ファイル名の指定がなければ、$Home/.ssh/config を想定する。
84 fn get_config_file(file_name: &Option<PathBuf>) -> Option<std::fs::File> {
85     let file_name = file_name.clone().or_else(|| {
86         home::home_dir().map(|p| {
87             let mut p = p;
88             p.push(".ssh/config");
89             p

```

```

90         })
91     });
92
93     file_name.and_then(|p| File::open(p).ok())
94 }
95
96 /// ログイン名を確定し、取得する。
97 /// ログイン名指定の優先順位は、1. -u 引数指定, 2.remote 引数, 3.ssh_config 指定, 4. 現在のユーザー名
98 fn get_username(opt: &Opt, params: &HostParams) -> Result<String> {
99     if let Some(n) = &opt.login_name {
100         Ok(n.clone())
101     } else if let Some(n) = &opt.remote.user {
102         Ok(n.clone())
103     } else if let Some(n) = &params.user {
104         Ok(n.clone())
105     } else if let Some(n) = users::get_current_username() {
106         n.to_str()
107             .map(|s| s.to_string())
108             .ok_or(anyhow!("Invalid login user name. -- {n:?}"))
109     } else {
110         Err(anyhow!("Could not obtain user name."))
111     }
112 }
113
114 /// 秘密キーファイルのパスを取得する
115 fn get_identity_file(opt: &Opt, host_params: &HostParams) -> Result<Option<PathBuf>> {
116     if let Some(n) = &opt.identity {
117         std::fs::File::open(n).with_context(|| {
118             format!(
119                 "Unable to access the secret key file specified by the \"-i\" option. [{:?}]",
120                 &n
121             )
122         })?;
123         Ok(Some(n.clone()))
124     } else {
125         let name = host_params.identity_file.as_ref().map(|p| p[0].clone());
126         if let Some(ref n) = name {
127             std::fs::File::open(n).with_context(|| {
128                 format!(
129                     "Unable to access the secret file specified by the ssh-config. [{:?}]",
130                     &n
131                 )
132             })?;
133         }
134         Ok(name)
135     }

```

```

136 }
137
138 /// リモートの ssh に接続し、セッションを生成する。
139 fn connect_ssh<A: std::net::ToSocketAddrs>(address: A) -> Result<Session> {
140     let tcp = TcpStream::connect(address).context("Failed to connect to TCP/IP.")?;
141     let mut ssh = Session::new().context("Failed to connect to ssh.")?;
142     ssh.set_tcp_stream(tcp);
143     ssh.handshake().context("Failed to handshake ssh.")?;
144     Ok(ssh)
145 }
146
147 /// ssh 認証を実施する。
148 fn userauth(sess: &Session, username: &str, identity: &Option<PathBuf>) -> Result<()> {
149     if user_auth_agent(sess, username).is_ok() {
150         return Ok(());
151     }
152     if let Some(f) = identity {
153         if user_auth_identity(sess, username, f).is_ok() {
154             return Ok(());
155         }
156     }
157     user_auth_password(sess, username)
158         .map_err(|_| anyhow!("All user authentication methods failed.))
159 }
160
161 /// agent 認証
162 fn user_auth_agent(sess: &Session, username: &str) -> Result<(), ssh2::Error> {
163     let ret = sess.userauth_agent(username);
164     if let Err(e) = &ret {
165         debug!("認証失敗 (agent)->{:?}", e);
166     };
167     ret
168 }
169
170 /// 公開キー認証
171 fn user_auth_identity(sess: &Session, username: &str, key_file: &Path) -> Result<(), String> {
172     let mut ret = sess.userauth_pubkey_file(username, None, key_file, None);
173     if ret.is_ok() {
174         return Ok(());
175     };
176     if let ssh2::ErrorCode::Session(-16) = ret.as_ref().unwrap_err().code() {
177         // error_code -16 ->
178         // LIBSSH2_ERROR_FILE_PUBLIC_KEY の取得失敗。多分、秘密キーのパスフレーズ
179         for _i in 0..3 {
180             let password = Password::new()
181                 .with_prompt("Enter the passphrase for the secret key.")

```

```

182         .allow_empty_password(true)
183         .interact()
184         .map_err(|e| e.to_string())?;
185     ret = sess.userauth_pubkey_file(username, None, key_file, Some(&password));
186     if ret.is_ok() {
187         return Ok(());
188     }
189     eprintln!("The passphrase is different.");
190 }
191 }
192 debug!("認証失敗 (pubkey)->{:?}", ret.as_ref().unwrap_err());
193 Err("公開キー認証失敗".to_string())
194 }
195
196 /// パスワード認証
197 fn user_auth_password(sess: &Session, username: &str) -> Result<(), String> {
198     for _i in 0..3 {
199         let password = Password::new()
200             .with_prompt("Enter your login password.")
201             .allow_empty_password(true)
202             .interact()
203             .map_err(|e| e.to_string())?;
204         let ret = sess.userauth_password(username, &password);
205         if ret.is_ok() {
206             return Ok(());
207         }
208         let ssh2::ErrorCode::Session(-18) = ret.as_ref().unwrap_err().code() else {
209             break;
210         };
211         // ssh2 エラーコード -18 ->
212         // LIBSSH2_ERROR_AUTHENTICATION_FAILED: パスワードが違うんでしょう。
213         eprintln!("The password is different.");
214         debug!("認証失敗 (password)->{:?}", ret.unwrap_err());
215     }
216     Err("パスワード認証失敗".to_string())
217 }

```

5 FUSE 接続オプション生成モジュール fuse_util.rs

```
1  /// FUSE パラメータ関係 ユーティリティ
2
3  use crate::cmdline_opt::Opt;
4  use anyhow::{ensure, Context, Result};
5  use ssh2::Session;
6  use std::env::current_dir;
7  use std::{
8      io::Read,
9      path::{Path, PathBuf},
10     str,
11 };
12
13 /// マウントポイントのフルパスを生成する
14 pub fn make_full_path<P: AsRef<Path>>(path: P) -> Result<PathBuf> {
15     if path.as_ref().is_absolute() {
16         Ok(path.as_ref().to_path_buf())
17     } else {
18         let mut full_path = current_dir().context("cannot access current directory.")?;
19         full_path.push(path);
20         Ok(full_path)
21     }
22 }
23
24 /// リモート接続先の path の生成
25 pub fn make_remote_path(opt: &Opt, session: &Session) -> Result<PathBuf> {
26     // パスの生成
27     const MSG_ERRORHOME: &str = "Fail to generate path name.";
28     let mut path = match opt.remote.path {
29         Some(ref p) => {
30             if p.is_absolute() {
31                 p.clone()
32             } else {
33                 let mut h = get_home_on_remote(session).context(MSG_ERRORHOME)?;
34                 h.push(p);
35                 h
36             }
37         }
38         None => get_home_on_remote(session).context(MSG_ERRORHOME)?,
39     };
40     // 生成したパスが実在するかを確認する
41     let sftp = session
42         .sftp()
43         .context("Connection to SFTP failed when checking for existence of a path.")?;
```

```

44     let file_stat = sftp
45         .stat(&path)
46         .with_context(|| format!("Cannot find path to connect to. path={:?}", &path))?;
47     ensure!(
48         file_stat.is_dir(),
49         "The path to connect to is not a directory."
50     );
51     // 生成したパスがシンボリックリンクのときは、リンク先を解決する
52     let file_stat = sftp
53         .lstat(&path)
54         .context("Failed to obtain the attributes of the destination directory.")?;
55     if file_stat.file_type().is_symlink() {
56         path = sftp
57             .readlink(&path)
58             .context("Failed to resolve symbolic link to connect to.")?;
59         if !path.is_absolute() {
60             let tmp = path;
61             path = get_home_on_remote(session)
62                 .context("Failed to complete the symbolic link to connect to.")?;
63             path.push(tmp);
64         };
65     };
66
67     Ok(path)
68 }
69
70 /// FUSEの接続時オプションを生成する
71 pub fn make_mount_option(cmd_opt: &Opt) -> Vec<fuser::MountOption> {
72     use fuser::MountOption;
73
74     let mut options = vec![MountOption::FSName("sshfs".to_string())];
75     options.push(MountOption::NoDev);
76     options.push(MountOption::DirSync);
77     options.push(MountOption::Sync);
78     match cmd_opt.readonly {
79         true => options.push(MountOption::RO),
80         false => options.push(MountOption::RW),
81     }
82     match cmd_opt.no_exec {
83         true => options.push(MountOption::NoExec),
84         false => options.push(MountOption::Exec),
85     }
86     match cmd_opt.no_atime {
87         true => options.push(MountOption::NoAtime),
88         false => options.push(MountOption::Atime),
89     }

```

```

90     options
91 }
92
93 /// ssh 接続先のカレントディレクトリを取得する
94 fn get_home_on_remote(session: &Session) -> Result<PathBuf> {
95     let mut channel = session
96         .channel_session()
97         .context("Fail to build ssh channel.")?;
98     channel
99         .exec("pwd")
100         .context("Fail to execute \"pwd\" command.")?;
101     let mut buf = Vec::<u8>::new();
102     channel
103         .read_to_end(&mut buf)
104         .context("Fail to get response for \"pwd\" command.")?;
105     channel.close().context("Fail to close ssh channel.")?;
106     str::from_utf8(&buf)
107         .context("The pwd result contains non-utf8 characters.")?
108         .trim()
109         .parse::<PathBuf>()
110         .context("Fail to build path name.")
111 }

```

6 ファイルシステムモジュール ssh_filesystem.rs

```
1  mod bi_hash_map;
2  mod file_handle;
3  mod inode;
4
5  use file_handle::Fhandles;
6  use inode::Inodes;
7
8  use anyhow::Context;
9  use fuser::{FileAttr, Filesystem, ReplyAttr, ReplyData, ReplyDirectory, ReplyEntry, Request};
10 use libc::ENOENT;
11 use log::{debug, error, warn};
12 use ssh2::{ErrorCode, OpenFlags, OpenType, Session, Sftp};
13 use std::{
14     ffi::OsStr,
15     io::{Read, Seek, SeekFrom, Write},
16     path::{Path, PathBuf},
17     time::{Duration, SystemTime, UNIX_EPOCH},
18 };
19
20 /// FUSE ファイルシステム実装
21 pub struct Sshfs {
22     _session: Session,
23     sftp: Sftp,
24     inodes: Inodes,
25     fhandles: Fhandles,
26     _top_path: PathBuf,
27 }
28
29 impl Sshfs {
30     pub fn new<P: AsRef<Path>>>(session: Session, path: P) -> anyhow::Result<Self> {
31         let mut inodes = Inodes::new();
32         let top_path: PathBuf = path.as_ref().into();
33         inodes.add(&top_path);
34         let sftp = session
35             .sftp()
36             .inspect_err(|_| {
37                 error!("Failed to create sftp from session.");
38             })
39             .context("Failed to create sftp from session.(Sshfs::new)");
40         debug!(
41             "[Sshfs::new] connect path: <{:?}>, inodes=<{:?}>",
42             &top_path, &inodes
43         );
44     }
45 }
```

```

44     Ok(Self {
45         _session: session,
46         sftp,
47         inodes,
48         fhndls: Fhandles::new(),
49         _top_path: top_path,
50     })
51 }
52
53 /// ssh2経由でファイルのステータスを取得する。
54 /// 副作用: 取得に成功した場合、inodesにパスを登録する。
55 fn getattr_from_ssh2(&mut self, path: &Path, uid: u32, gid: u32) -> Result<FileAttr, Error> {
56     let attr_ssh2 = self.sftp.lstat(path)?;
57     let kind = Self::conv_file_kind_ssh2fuser(&attr_ssh2.file_type())?;
58     let ino = self.inodes.add(path);
59     Ok(FileAttr {
60         ino,
61         size: attr_ssh2.size.unwrap_or(0),
62         blocks: attr_ssh2.size.unwrap_or(0) / 512 + 1,
63         atime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.atime.unwrap_or(0)),
64         mtime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.mtime.unwrap_or(0)),
65         ctime: UNIX_EPOCH + Duration::from_secs(attr_ssh2.mtime.unwrap_or(0)),
66         crtime: UNIX_EPOCH,
67         kind,
68         perm: attr_ssh2.perm.unwrap_or(0o666) as u16,
69         nlink: 1,
70         uid,
71         gid,
72         rdev: 0,
73         blksize: 512,
74         flags: 0,
75     })
76 }
77
78 fn conv_file_kind_ssh2fuser(filetype: &ssh2::FileType) -> Result<fuser::FileType, Error> {
79     match filetype {
80         ssh2::FileType::NamedPipe => Ok(fuser::FileType::NamedPipe),
81         ssh2::FileType::CharDevice => Ok(fuser::FileType::CharDevice),
82         ssh2::FileType::BlockDevice => Ok(fuser::FileType::BlockDevice),
83         ssh2::FileType::Directory => Ok(fuser::FileType::Directory),
84         ssh2::FileType::RegularFile => Ok(fuser::FileType::RegularFile),
85         ssh2::FileType::Symlink => Ok(fuser::FileType::Symlink),
86         ssh2::FileType::Socket => Ok(fuser::FileType::Socket),
87         ssh2::FileType::Other(_) => Err(Error(libc::EBADF)),
88     }
89 }

```

```

90
91     fn conv_timeornow2systemtime(time: &fuser::TimeOrNow) -> SystemTime {
92         match time {
93             fuser::TimeOrNow::SpecificTime(t) => *t,
94             fuser::TimeOrNow::Now => SystemTime::now(),
95         }
96     }
97 }
98
99 impl Filesystem for Sshfs {
100     fn lookup(&mut self, req: &Request, parent: u64, name: &OsStr, reply: ReplyEntry) {
101         debug!("[lookup] start: parent={}, name={:?}", parent, name);
102         let Some(mut path) = self.inodes.get_path(parent) else {
103             debug!("[lookup] 親ディレクトリの検索に失敗 inode={}", parent);
104             reply.error(ENOENT);
105             return;
106         };
107         path.push(Path::new(name));
108         match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
109             Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
110             Err(e) => {
111                 reply.error(e.0);
112             }
113         };
114     }
115
116     fn getattr(&mut self, req: &Request, ino: u64, _fh: Option<u64>, reply: ReplyAttr) {
117         debug!("[getattr] start: ino={}", ino);
118         let Some(path) = self.inodes.get_path(ino) else {
119             debug!("[getattr] path 取得失敗: inode={}", ino);
120             reply.error(ENOENT);
121             return;
122         };
123         match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
124             Ok(attr) => {
125                 //debug!("[getattr] retrun attr: {:?}", &attr);
126                 reply.attr(&Duration::from_secs(1), &attr);
127             }
128             Err(e) => {
129                 warn!("[getattr] getattr_from_ssh2 エラー: {:?}", &e);
130                 reply.error(e.0)
131             }
132         };
133     }
134
135     fn readdir(

```

```

136     &mut self,
137     _req: &Request,
138     ino: u64,
139     _fh: u64,
140     offset: i64,
141     mut reply: ReplyDirectory,
142 ) {
143     debug!("[readdir] start: ino={}, offset={}", ino, offset);
144     let Some(path) = self.inodes.get_path(ino) else {
145         reply.error(libc::ENOENT);
146         return;
147     };
148     match self.sftp.readdir(&path) {
149         Ok(mut dir) => {
150             let cur_file_attr = ssh2::FileStat {
151                 size: None,
152                 uid: None,
153                 gid: None,
154                 perm: Some(libc::S_IFDIR),
155                 atime: None,
156                 mtime: None,
157             }; // "." ".."の解決用。 attr ディレクトリであることを示す。
158             dir.insert(0, (Path::new("..").into(), cur_file_attr.clone()));
159             dir.insert(0, (Path::new(".").into(), cur_file_attr));
160             let mut i = offset + 1;
161             for f in dir.iter().skip(offset as usize) {
162                 let ino = if f.0 == Path::new("..") || f.0 == Path::new(".") {
163                     1
164                 } else {
165                     self.inodes.add(&f.0)
166                 };
167                 let name = match f.0.file_name() {
168                     Some(n) => n,
169                     None => f.0.as_os_str(),
170                 };
171                 let filetype = &f.1.file_type();
172                 let filetype = match Self::conv_file_kind_ssh2fuser(filetype) {
173                     Ok(t) => t,
174                     Err(e) => {
175                         warn!(
176                             "[readdir] ファイルタイプ解析失敗: inode={}, name={:?}",
177                             ino, name
178                         );
179                         reply.error(e.0);
180                         return;
181                     }
182                 };

```

```

182         };
183         if reply.add(ino, i, filetype, name) {
184             break;
185         }
186         i += 1;
187     }
188     reply.ok();
189 }
190 Err(e) => {
191     warn!("[readdir] ssh2::readdir 内でエラー発生 -- {:?}", e);
192     reply.error(Error::from(e).0);
193 }
194 };
195 }
196
197 fn readlink(&mut self, _req: &Request<'_>, ino: u64, reply: ReplyData) {
198     debug!("[readlink] start: ino={}", ino);
199     let Some(path) = self.inodes.get_path(ino) else {
200         error!("[readlink] 親ディレクトリの検索に失敗 {ino}");
201         reply.error(libc::ENOENT);
202         return;
203     };
204     match self.sftp.readlink(&path) {
205         Ok(p) => {
206             //debug!("[readlink] ret_path => {:?}", &p);
207             reply.data(p.as_os_str().to_string_lossy().as_bytes());
208         }
209         Err(e) => {
210             //debug!("[readlink] ssh2::readlink error => {e:?}");
211             reply.error(Error::from(e).0);
212         }
213     }
214 }
215
216 fn open(&mut self, _req: &Request<'_>, ino: u64, flags: i32, reply: fuser::ReplyOpen) {
217     debug!("[open] start: ino={}, flags={:x}", ino, flags);
218     let Some(file_name) = self.inodes.get_path(ino) else {
219         reply.error(libc::ENOENT);
220         return;
221     };
222
223     let mut flags_ssh2 = OpenFlags::empty();
224     if flags & libc::O_WRONLY != 0 {
225         flags_ssh2.insert(OpenFlags::WRITE);
226     } else if flags & libc::O_RDWR != 0 {
227         flags_ssh2.insert(OpenFlags::READ);

```

```

228         flags_ssh2.insert(OpenFlags::WRITE);
229     } else {
230         flags_ssh2.insert(OpenFlags::READ);
231     }
232     if flags & libc::O_APPEND != 0 {
233         flags_ssh2.insert(OpenFlags::APPEND);
234     }
235     if flags & libc::O_CREAT != 0 {
236         flags_ssh2.insert(OpenFlags::CREATE);
237     }
238     if flags & libc::O_TRUNC != 0 {
239         flags_ssh2.insert(OpenFlags::TRUNCATE);
240     }
241     if flags & libc::O_EXCL != 0 {
242         flags_ssh2.insert(OpenFlags::EXCLUSIVE);
243     }
244
245     debug!(
246         "[open] filename='{:?}', openflag = {:?}, bit = {:x}",
247         &file_name,
248         &flags_ssh2,
249         flags_ssh2.bits()
250     );
251     match self
252         .sftp
253         .open_mode(&file_name, flags_ssh2, 0o777, ssh2::OpenType::File)
254     {
255         Ok(file) => {
256             let fh = self.fhndls.add_file(file);
257             reply.opened(fh, flags as u32);
258         }
259         Err(e) => {
260             log::error!(
261                 "file-open error: filename='{:?}', mode={:?}, err={}",
262                 &file_name,
263                 &flags_ssh2,
264                 &e
265             );
266             reply.error(Error::from(e).0);
267         }
268     }
269 }
270
271 fn release(
272     &mut self,
273     _req: &Request<'_,

```

```

274     _ino: u64,
275     fh: u64,
276     _flags: i32,
277     _lock_owner: Option<u64>,
278     _flush: bool,
279     reply: fuser::ReplyEmpty,
280 ) {
281     debug!("[release] start: fh={}", fh);
282     self.fhandles.del_file(fh);
283     reply.ok();
284 }
285
286 fn read(
287     &mut self,
288     _req: &Request,
289     _ino: u64,
290     fh: u64,
291     offset: i64,
292     size: u32,
293     _flags: i32,
294     _lock_owner: Option<u64>,
295     reply: ReplyData,
296 ) {
297     debug!("[read] start: fh={}, offset={}, size={}", fh, offset, size);
298     let Some(file_mutex) = self.fhandles.get_file(fh) else {
299         reply.error(libc::EBADF);
300         return;
301     };
302     let mut file = file_mutex.lock().unwrap();
303     // 注釈: このデータの出所である Fhandles 構造体内のデータに毒化があるということは、
304     // システム全域の他のファイルハンドルの正当性も保証できないことを意味する。
305     // ファイル操作を失敗させることより、システム全体を落とすことが正しい選択と思われる。
306     // よって、lock().unwrap() とする。write() 関数他においても同様。
307
308     if let Err(e) = file.seek(std::io::SeekFrom::Start(offset as u64)) {
309         reply.error(Error::from(e).0);
310         return;
311     }
312     let mut buff = vec![0; size as usize];
313     let mut read_size: usize = 0;
314     while read_size < size as usize {
315         match file.read(&mut buff[read_size..]) {
316             Ok(s) => {
317                 if s == 0 {
318                     break;
319                 }

```

```

320         read_size += s;
321     }
322     Err(e) => {
323         reply.error(Error::from(e).0);
324         return;
325     }
326 }
327 }
328 buff.resize(read_size, 0u8);
329 reply.data(&buff);
330 }
331
332 fn write(
333     &mut self,
334     _req: &Request<'_>,
335     _ino: u64,
336     fh: u64,
337     offset: i64,
338     data: &[u8],
339     _write_flags: u32,
340     _flags: i32,
341     _lock_owner: Option<u64>,
342     reply: fuser::ReplyWrite,
343 ) {
344     debug!(
345         "[write] start: fh={}, offset={}, size={}",
346         fh,
347         offset,
348         data.len()
349     );
350     let Some(file_mutex) = self.fhandles.get_file(fh) else {
351         reply.error(libc::EBADF);
352         return;
353     };
354     let file = &mut file_mutex.lock().unwrap();
355
356     if let Err(e) = file.seek(std::io::SeekFrom::Start(offset as u64)) {
357         reply.error(Error::from(e).0);
358         return;
359     }
360     let mut buf = data;
361     while !buf.is_empty() {
362         let cnt = match file.write(buf) {
363             Ok(cnt) => cnt,
364             Err(e) => {
365                 reply.error(Error::from(e).0);

```

```

366         return;
367     }
368 };
369     buf = &buf[cnt..];
370 }
371     reply.written(data.len() as u32);
372 }
373
374 fn lseek(
375     &mut self,
376     _req: &Request<'_,>,
377     _ino: u64,
378     fh: u64,
379     offset: i64,
380     whence: i32,
381     reply: fuser::ReplyLseek,
382 ) {
383     debug!(
384         "[lseek] start: fh={}, offset={}, whence={}",
385         fh, offset, whence
386     );
387     let seek_from = match whence {
388         libc::SEEK_SET => {
389             if offset >= 0 {
390                 SeekFrom::Start(offset as u64)
391             } else {
392                 reply.error(libc::EINVAL);
393                 return;
394             }
395         }
396         libc::SEEK_CUR => SeekFrom::Current(offset),
397         libc::SEEK_END => SeekFrom::End(offset),
398         _ => {
399             reply.error(libc::EINVAL);
400             return;
401         }
402     };
403     let Some(file_mutex) = self.fhandls.get_file(fh) else {
404         reply.error(libc::EBADF);
405         return;
406     };
407     let file = &mut file_mutex.lock().unwrap();
408     let pos = match file.seek(seek_from) {
409         Ok(p) => p,
410         Err(e) => {
411             reply.error(Error::from(e).0);

```

```

412         return;
413     }
414 };
415 reply.offset(pos as i64);
416 }
417
418 fn mknod(
419     &mut self,
420     req: &Request<'_,>,
421     parent: u64,
422     name: &OsStr,
423     mode: u32,
424     umask: u32,
425     _rdev: u32,
426     reply: ReplyEntry,
427 ) {
428     debug!(
429         "[mknod] start: parent={}, name={:?}, mode={:o}, umask={:o}",
430         parent, name, mode, umask
431     );
432     if mode & libc::S_IFMT != libc::S_IFREG {
433         reply.error(libc::EPERM);
434         return;
435     }
436     let mode = mode & (!umask | libc::S_IFMT);
437     let Some(mut new_name) = self.inodes.get_path(parent) else {
438         reply.error(libc::ENOENT);
439         return;
440     };
441     new_name.push(name);
442     if let Err(e) =
443         self.sftp
444             .open_mode(&new_name, OpenFlags::CREATE, mode as i32, OpenType::File)
445     {
446         reply.error(Error::from(e).0);
447         return;
448     }
449     let new_attr = match self.getattr_from_ssh2(&new_name, req.uid(), req.gid()) {
450         Ok(a) => a,
451         Err(e) => {
452             reply.error(e.0);
453             return;
454         }
455     };
456     reply.entry(&Duration::from_secs(1), &new_attr, 0);
457 }

```

```

458
459 fn unlink(&mut self, _req: &Request<'_,>, parent: u64, name: &OsStr, reply: fuser::ReplyEmpty) {
460     debug!("[unlink] start: parent={}, name={:?}", parent, name);
461     let Some(mut path) = self.inodes.get_path(parent) else {
462         reply.error(libc::ENOENT);
463         return;
464     };
465     path.push(name);
466     match self.sftp.unlink(&path) {
467         Ok(_) => {
468             self.inodes.del_inode_with_path(&path);
469             reply.ok();
470         }
471         Err(e) => reply.error(Error::from(e).0),
472     }
473 }
474
475 fn mkdir(
476     &mut self,
477     req: &Request<'_,>,
478     parent: u64,
479     name: &OsStr,
480     mode: u32,
481     umask: u32,
482     reply: ReplyEntry,
483 ) {
484     debug!(
485         "[mkdir] start: parent={}, name={:?}", mode={:o}, umask={:o}",
486         parent, name, mode, umask
487     );
488     let Some(mut path) = self.inodes.get_path(parent) else {
489         reply.error(libc::ENOENT);
490         return;
491     };
492     path.push(name);
493
494     let mode = (mode & (!umask) & 0o777) as i32;
495
496     match self.sftp.mkdir(&path, mode) {
497         Ok(_) => match self.getattr_from_ssh2(&path, req.uid(), req.gid()) {
498             Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
499             Err(e) => reply.error(e.0),
500         },
501         Err(e) => reply.error(Error::from(e).0),
502     }
503 }

```

```

504
505 fn rmdir(&mut self, _req: &Request<'_, parent: u64, name: &OsStr, reply: fuser::ReplyEmpty) {
506     debug!("[rmdir] start: parent={}, name={:?}", parent, name);
507     let Some(mut path) = self.inodes.get_path(parent) else {
508         reply.error(libc::ENOENT);
509         return;
510     };
511     path.push(name);
512     match self.sftp.rmdir(&path) {
513         Ok(_) => {
514             self.inodes.del_inode_with_path(&path);
515             reply.ok()
516         }
517         Err(e) => {
518             if e.code() == ErrorCode::Session(-31) {
519                 // ssh2 ライブラリの返すエラーが妙。置換しておく。
520                 reply.error(libc::ENOTEMPTY);
521             } else {
522                 reply.error(Error::from(e).0)
523             }
524         }
525     }
526 }
527
528 fn symlink(
529     &mut self,
530     req: &Request<'_,
531     parent: u64,
532     name: &OsStr,
533     link: &Path,
534     reply: ReplyEntry,
535 ) {
536     debug!(
537         "[symlink] start: parent={}, name={:?}", link={:?}",
538         parent, name, link
539     );
540     let Some(mut target) = self.inodes.get_path(parent) else {
541         reply.error(libc::ENOENT);
542         return;
543     };
544     target.push(name);
545     match self.sftp.symlink(link, &target) {
546         Ok(_) => match self.getattr_from_ssh2(&target, req.uid(), req.gid()) {
547             Ok(attr) => reply.entry(&Duration::from_secs(1), &attr, 0),
548             Err(e) => reply.error(e.0),
549         },

```

```

550         Err(e) => reply.error(Error::from(e).0),
551     }
552 }
553
554 fn setattr(
555     &mut self,
556     req: &Request<'_>,
557     ino: u64,
558     mode: Option<u32>,
559     _uid: Option<u32>,
560     _gid: Option<u32>,
561     size: Option<u64>,
562     atime: Option<fuser::TimeOrNow>,
563     mtime: Option<fuser::TimeOrNow>,
564     _ctime: Option<std::time::SystemTime>,
565     _fh: Option<u64>,
566     _ctime: Option<std::time::SystemTime>,
567     _chmtime: Option<std::time::SystemTime>,
568     _bkuptime: Option<std::time::SystemTime>,
569     _flags: Option<u32>,
570     reply: ReplyAttr,
571 ) {
572     debug!(
573         "[setattr] start: ino={}, mode={:?}", size={:?}, atime={:?}", mtime={:?}",
574         ino, mode, size, atime, mtime
575     );
576     let stat = ssh2::FileStat {
577         size,
578         uid: None,
579         gid: None,
580         perm: mode,
581         atime: atime.map(|t| {
582             Self::conv_timeornow2systemtime(&t)
583                 .duration_since(UNIX_EPOCH)
584                 .unwrap_or_default()
585                 .as_secs()
586         }),
587         mtime: mtime.map(|t| {
588             Self::conv_timeornow2systemtime(&t)
589                 .duration_since(UNIX_EPOCH)
590                 .unwrap_or_default()
591                 .as_secs()
592         }),
593     };
594     let Some(filename) = self.inodes.get_path(ino) else {
595         reply.error(ENOENT);

```

```

596         return;
597     };
598     match self.sftp.setstat(&filename, stat) {
599         Ok(_) => {
600             let stat = self.getattr_from_ssh2(&filename, req.uid(), req.gid());
601             match stat {
602                 Ok(s) => reply.attr(&Duration::from_secs(1), &s),
603                 Err(e) => reply.error(e.0),
604             }
605         }
606         Err(e) => reply.error(Error::from(e).0),
607     }
608 }
609
610 fn rename(
611     &mut self,
612     _req: &Request<'_>,
613     parent: u64,
614     name: &OsStr,
615     newparent: u64,
616     newname: &OsStr,
617     flags: u32,
618     reply: fuser::ReplyEmpty,
619 ) {
620     debug!(
621         "[rename] start: parent={}, name={:?}, newparent={}, newname={:?}, flags={}",
622         parent, name, newparent, newname, flags
623     );
624     let Some(mut old_path) = self.inodes.get_path(parent) else {
625         reply.error(libc::ENOENT);
626         return;
627     };
628     old_path.push(name);
629
630     let Some(mut new_path) = self.inodes.get_path(newparent) else {
631         reply.error(libc::ENOENT);
632         return;
633     };
634     new_path.push(newname);
635
636     let mut rename_flag = ssh2::RenameFlags::NATIVE;
637     if flags & libc::RENAME_EXCHANGE != 0 {
638         rename_flag.insert(ssh2::RenameFlags::ATOMIC);
639     }
640     if flags & libc::RENAME_NOREPLACE == 0 {
641         // rename の OVERWRITE が効いてない。手動で消す。

```

```

642         if let Ok(stat) = self.sftp.lstat(&new_path) {
643             if stat.is_dir() {
644                 if let Err(e) = self.sftp.rmdir(&new_path) {
645                     reply.error(Error::from(e).0);
646                     return;
647                 }
648             } else if let Err(e) = self.sftp.unlink(&new_path) {
649                 reply.error(Error::from(e).0);
650                 return;
651             }
652             self.inodes.del_inode_with_path(&new_path);
653         }
654     }
655
656     match self.sftp.rename(&old_path, &new_path, Some(rename_flag)) {
657         Ok(_) => {
658             self.inodes.rename(&old_path, &new_path);
659             reply.ok();
660         }
661         Err(e) => reply.error(Error::from(e).0),
662     }
663 }
664 }
665
666 #[derive(Debug, Clone, Copy)]
667 struct Error(i32);
668
669 impl From<ssh2::Error> for Error {
670     fn from(value: ssh2::Error) -> Self {
671         let eno = match value.code() {
672             ssh2::ErrorCode::Session(_) => libc::ENXIO,
673             ssh2::ErrorCode::SFTP(i) => match i {
674                 // libssh2のlibssh2_sftp.hにて定義されている。
675                 2 => libc::ENOENT,           // NO_SUCH_FILE
676                 3 => libc::EACCES,          // permission_denied
677                 4 => libc::EIO,             // failure
678                 5 => libc::ENODEV,          // bad message
679                 6 => libc::ENXIO,           // no connection
680                 7 => libc::ENETDOWN,        // connection lost
681                 8 => libc::ENODEV,          // unsported
682                 9 => libc::EBADF,           // invalid handle
683                 10 => libc::ENOENT,         //no such path
684                 11 => libc::EEXIST,         // file already exists
685                 12 => libc::EACCES,         // write protected
686                 13 => libc::ENXIO,          // no media
687                 14 => libc::ENOSPC,         // no space on filesystem

```

```

688         15 => libc::EDQUOT,          // quota exceeded
689         16 => libc::ENODEV,          // unknown principal
690         17 => libc::ENOLCK,           // lock conflict
691         18 => libc::ENOTEMPTY,        // dir not empty
692         19 => libc::ENOTDIR,          // not a directory
693         20 => libc::ENAMETOOLONG,     // invalid file name
694         21 => libc::ELOOP,            // link loop
695         _ => {
696             error!("An unknown error occurred during SSH2.{}", i);
697             libc::EIO
698         }
699     },
700 };
701 Self(eno)
702 }
703 }
704
705 impl From<std::io::Error> for Error {
706     fn from(value: std::io::Error) -> Self {
707         use std::io::ErrorKind::*;
708         let eno = match value.kind() {
709             NotFound => libc::ENOENT,
710             PermissionDenied => libc::EACCES,
711             ConnectionRefused => libc::ECONNREFUSED,
712             ConnectionReset => libc::ECONNRESET,
713             ConnectionAborted => libc::ECONNABORTED,
714             NotConnected => libc::ENOTCONN,
715             AddrInUse => libc::EADDRINUSE,
716             AddrNotAvailable => libc::EADDRNOTAVAIL,
717             BrokenPipe => libc::EPIPE,
718             AlreadyExists => libc::EEXIST,
719             WouldBlock => libc::EWOULDBLOCK,
720             InvalidInput => libc::EINVAL,
721             InvalidData => libc::EILSEQ,
722             TimedOut => libc::ETIMEDOUT,
723             WriteZero => libc::EIO,
724             Interrupted => libc::EINTR,
725             Unsupported => libc::ENOTSUP,
726             UnexpectedEof => libc::EOF,
727             OutOfMemory => libc::ENOMEM,
728             _ => {
729                 error!(
730                     "An unknown error occurred during std::io.{}",
731                     value.kind()
732                 );
733                 libc::EIO

```

```
734         }  
735     };  
736     Self(eno)  
737 }  
738 }
```

7 ファイルハンドル管理モジュール `ssh_filesystem/file_handle.rs`

```
1  //! ファイルハンドル管理モジュール
2
3  use std::collections::HashMap;
4  use std::sync::{
5      atomic::{AtomicU64, Ordering},
6      Arc, Mutex,
7  };
8
9  /// ファイルハンドル管理構造体
10 pub(super) struct Fhandles {
11     list: Mutex<HashMap<u64, Arc<Mutex<ssh2::File>>>>,
12     next_handle: AtomicU64,
13 }
14
15 impl Fhandles {
16     pub(super) fn new() -> Self {
17         Self {
18             list: Mutex::new(HashMap::new()),
19             next_handle: AtomicU64::new(0),
20         }
21     }
22
23     pub(super) fn add_file(&mut self, file: ssh2::File) -> u64 {
24         let handle = self.next_handle.fetch_add(1, Ordering::AcqRel);
25         self.list
26             .lock()
27             .unwrap()
28             .insert(handle, Arc::new(Mutex::new(file)));
29         handle
30         // 注釈: このリストが毒化されたら、もはや、全システムにわたり、ファイル操作の正当性を保証でき
31         ↳ ない。
32         // プログラムとしてできることは即座にシステムを落とすことだけである。
33         // よって、このモジュール内において、lock().unwrap() とする。
34     }
35
36     pub(super) fn get_file(&self, fh: u64) -> Option<Arc<Mutex<ssh2::File>>> {
37         self.list.lock().unwrap().get(&fh).cloned()
38     }
39
40     pub(super) fn del_file(&mut self, fh: u64) {
41         self.list.lock().unwrap().remove(&fh); // 戻り値は捨てる。この時点でファイルはクローズ。
42     }
43 }
```

8 Inode 管理モジュール ssh_filesystem/inode.rs

```
1  /// Inode 管理モジュール
2
3  use super::bi_hash_map::BiHashMap;
4
5  use std::path::{Path, PathBuf};
6  use std::sync::{
7      atomic::{AtomicU64, Ordering},
8      Mutex,
9  };
10
11 /// Inode 管理構造体
12 #[derive(Debug, Default)]
13 pub(super) struct Inodes {
14     list: Mutex<BiHashMap<u64, PathBuf>>,
15     next_inode: AtomicU64,
16 }
17
18 impl Inodes {
19     /// Inodes を生成する
20     pub(super) fn new() -> Self {
21         Self {
22             list: Mutex::new(BiHashMap::new()),
23             next_inode: AtomicU64::new(1),
24         }
25     }
26
27     /// path で指定された inode を生成し、登録する。
28     /// すでに path の登録が存在する場合、追加はせず、登録済みの inode を返す。
29     /// 初めて、この関数が呼ばれるときは、ファイルシステムにおけるルートであり、inode 番号 1 が割り当てら
30         れる。
31     pub(super) fn add<P: AsRef<Path>>(&mut self, path: P) -> u64 {
32         let mut list_guard = self.list.lock().unwrap();
33         // 注釈: このリストが毒化されたら、もはや、全システムにわたり、inode 管理の正当性を保証できない。
34         // 最善の方法が、即時システムを落とすことである。
35         // 以下、このモジュール全体に共通。
36         let path = PathBuf::from(path.as_ref());
37         match list_guard.get_left(&path) {
38             Some(i) => *i,
39             None => {
40                 let inode = self.next_inode.fetch_add(1, Ordering::AcqRel);
41                 if list_guard.insert_no_overwrite(inode, path.clone()).is_err() {
42                     unreachable!("Unexpected duplicate inode {} or path {:?}", inode, path);
43                     // 既に重複がチェックされているので、ありえない。
44                 }
45             }
46         }
47     }
48 }
```

```

43         }
44         inode
45     }
46 }
47
48
49 /// path から inode を取得する
50 /// 主用途が消滅したが、将来のために残しておく
51 #[allow(dead_code)]
52 pub(super) fn get_inode<P: AsRef<Path>>(&self, path: P) -> Option<u64> {
53     let path = PathBuf::from(path.as_ref());
54     self.list.lock().unwrap().get_left(&path).copied()
55 }
56
57 /// inode から path を取得する
58 pub(super) fn get_path(&self, inode: u64) -> Option<PathBuf> {
59     self.list.lock().unwrap().get_right(&inode).cloned()
60 }
61
62 /// inodes から、inode の登録を削除する
63 /// (主用途がなくなっちゃったけど、将来のために残しておく)
64 #[allow(dead_code)]
65 pub(super) fn del_inode(&mut self, inode: u64) -> Option<u64> {
66     self.list.lock().unwrap().remove_left(&inode).map(|_| inode)
67 }
68
69 /// path 名から iNode の登録を削除する
70 pub(super) fn del_inode_with_path<P: AsRef<Path>>(&mut self, path: P) -> Option<u64> {
71     let path = PathBuf::from(path.as_ref());
72     self.list.lock().unwrap().remove_right(&path)
73 }
74
75 /// 登録されている inode の path を変更する。
76 /// old_path が存在しなければ、なにもしない。
77 pub(super) fn rename<P: AsRef<Path>>(&mut self, old_path: P, new_path: P) {
78     let old_path = PathBuf::from(old_path.as_ref());
79     let new_path = PathBuf::from(new_path.as_ref());
80     let mut list_guard = self.list.lock().unwrap();
81     let Some(ino) = list_guard.get_left(&old_path).copied() else {
82         return;
83     };
84     list_guard.remove_left(&ino);
85     list_guard.insert(ino, new_path);
86 }
87 }
88

```

```

89  #[cfg(test)]
90  mod inode_test {
91      use super::Inodes;
92      use std::path::Path;
93
94      #[test]
95      fn inode_add_test() {
96          let mut inodes = Inodes::new();
97          assert_eq!(inodes.add(""), 1);
98          assert_eq!(inodes.add(Path::new("test")), 2);
99          assert_eq!(inodes.add(Path::new("")), 1);
100         assert_eq!(inodes.add(Path::new("test")), 2);
101         assert_eq!(inodes.add(Path::new("test3")), 3);
102         assert_eq!(inodes.add(Path::new("/test")), 4);
103         assert_eq!(inodes.add(Path::new("test/")), 2);
104     }
105
106     fn make_inodes() -> Inodes {
107         let mut inodes = Inodes::new();
108         inodes.add(Path::new(""));
109         inodes.add(Path::new("test"));
110         inodes.add(Path::new("test2"));
111         inodes.add(Path::new("test3/"));
112         inodes
113     }
114
115     #[test]
116     fn inodes_get_inode_test() {
117         let inodes = make_inodes();
118         assert_eq!(inodes.get_inode(Path::new("")), Some(1));
119         assert_eq!(inodes.get_inode(Path::new("test4")), None);
120         assert_eq!(inodes.get_inode(Path::new("/test")), None);
121         assert_eq!(inodes.get_inode(Path::new("test3")), Some(4));
122     }
123
124     #[test]
125     fn inodes_get_path_test() {
126         let inodes = make_inodes();
127         assert_eq!(inodes.get_path(1), Some(Path::new("").into()));
128         assert_eq!(inodes.get_path(3), Some(Path::new("test2").into()));
129         assert_eq!(inodes.get_path(5), None);
130         assert_eq!(inodes.get_path(3), Some(Path::new("test2/").into()));
131     }
132
133     #[test]
134     fn inodes_rename() {

```

```

135     let mut inodes = make_inodes();
136     let old = Path::new("test2");
137     let new = Path::new("new_test");
138     let ino = inodes.get_inode(old).unwrap();
139     inodes.rename(old, new);
140     assert_eq!(inodes.get_path(ino), Some(new.into()));
141
142     let mut inodes = make_inodes();
143     let inodes2 = make_inodes();
144     inodes.rename(Path::new("nai"), Path::new("kawattenai"));
145     assert_eq!(*inodes.list.lock().unwrap(), *inodes2.list.lock().unwrap());
146 }
147 }

```

9 双方向ハッシュマップモジュール `ssh_filesystem/bi_hash_map.rs`

```
1  ///! bidirectional hash map
2  ///! 双方向ハッシュマップ
3
4  use std::{collections::HashMap, hash::Hash, sync::Arc};
5
6  /// 双方向ハッシュマップ
7  #[derive(Debug, Default, PartialEq, Eq)]
8  pub(super) struct BiHashMap<L, R>
9  where
10     L: Hash + Eq + Clone,
11     R: Hash + Eq + Clone,
12  {
13     left: HashMap<Arc<L>, Arc<R>>,
14     right: HashMap<Arc<R>, Arc<L>>,
15  }
16
17  impl<L, R> BiHashMap<L, R>
18  where
19     L: Hash + Eq + Clone,
20     R: Hash + Eq + Clone,
21  {
22     /// 新しい双方向ハッシュマップを生成する
23     pub fn new() -> Self {
24         BiHashMap {
25             right: HashMap::new(),
26             left: HashMap::new(),
27         }
28     }
29
30     /// チェック無しで挿入する。
31     /// この時点で与えられる引数は、R,Lのいずれも既存のキーと重複しないことが保証されている必要がある。
32     fn insert_no_check(&mut self, left: L, right: R) {
33         let right = Arc::new(right);
34         let left = Arc::new(left);
35         self.right.insert(right.clone(), left.clone());
36         self.left.insert(left, right);
37     }
38
39     /// マップに新しい要素を挿入する。
40     /// 既存のキーと重複する場合、対応する値を上書きし、OverwriteResultで通知する。
41     pub fn insert(&mut self, left: L, right: R) -> OverwriteResult<L, R> {
42         let old_left = self.right.remove(&right);
43         let old_right = self.left.remove(&left);
```

```

44     let result = match (old_left, old_right) {
45         (None, None) => OverwriteResult::NoOverwrite,
46         (Some(old_l), None) => {
47             self.left.remove(old_l.as_ref());
48             OverwriteResult::OverwriteLeft((*old_l).clone())
49         }
50         (None, Some(old_r)) => {
51             self.right.remove(old_r.as_ref());
52             OverwriteResult::OverwriteRight((*old_r).clone())
53         }
54         (Some(old_l), Some(old_r)) => {
55             self.left.remove(old_l.as_ref());
56             self.right.remove(old_r.as_ref());
57             OverwriteResult::OverwriteBoth(
58                 (left.clone(), (*old_r).clone()),
59                 ((*old_l).clone(), right.clone()),
60             )
61         }
62     };
63     self.insert_no_check(left, right);
64     result
65 }
66
67 /// マップに新しい値を挿入する
68 /// 既存のキーが存在する場合は、エラーを返す。
69 pub fn insert_no_overwrite(&mut self, left: L, right: R) -> Result<(), ()> {
70     if self.contains_left(&left) || self.contains_right(&right) {
71         return Err(());
72     }
73     self.insert_no_check(left, right);
74     Ok(())
75 }
76
77 /// 左側のキーから右側の値を取得する
78 /// 存在しない場合は None を返す
79 pub fn get_right(&self, left: &L) -> Option<&R> {
80     self.left.get(left).map(|arc_r| arc_r.as_ref())
81 }
82
83 /// 右側のキーから左側の値を取得する
84 /// 存在しない場合は None を返す
85 pub fn get_left(&self, right: &R) -> Option<&L> {
86     self.right.get(right).map(|arc_l| arc_l.as_ref())
87 }
88
89 /// 左側のキーが存在するかどうかを返す

```

```

90     /// 存在する場合は true、存在しない場合は false を返す
91     pub fn contains_left(&self, left: &L) -> bool {
92         self.left.contains_key(left)
93     }
94
95     /// 右側のキーが存在するかどうかを返す
96     /// 存在する場合は true、存在しない場合は false を返す
97     pub fn contains_right(&self, right: &R) -> bool {
98         self.right.contains_key(right)
99     }
100
101     /// 左側の値から、リストの項目を削除する
102     /// 存在しない場合は、なにもしない。
103     /// 以前の値を返す。(存在しない場合は None)
104     pub fn remove_left(&mut self, left: &L) -> Option<R> {
105         let result = self.left.remove(left).map(|arc_r| (*arc_r).clone());
106         if let Some(right) = result.as_ref() {
107             self.right.remove(right);
108         };
109         result
110     }
111
112     /// 右側の値から、リストの項目を削除する
113     /// 存在しない場合は、なにもしない。
114     /// 以前の値を返す。(存在しない場合は None)
115     pub fn remove_right(&mut self, right: &R) -> Option<L> {
116         let result = self.right.remove(right).map(|arc_l| (*arc_l).clone());
117         if let Some(left) = result.as_ref() {
118             self.left.remove(left);
119         };
120         result
121     }
122 }
123
124 /// 挿入時の上書き結果
125 #[derive(Debug, PartialEq, Eq)]
126 pub enum OverwriteResult<L, R> {
127     NoOverwrite,
128     OverwriteRight(R),
129     OverwriteLeft(L),
130     OverwriteBoth((L, R), (L, R)),
131 }
132
133 #[cfg(test)]
134 mod tests {
135     use super::*;

```

```

136
137 #[test]
138 /// 単純な挿入と、値の取得のテスト
139 fn tanjyunna_insert() {
140     let mut bimap = BiHashMap::new();
141     assert_eq!(bimap.insert(1, "a"), OverwriteResult::NoOverwrite);
142     assert_eq!(bimap.get_right(&1), Some(&"a"));
143     assert_eq!(bimap.get_left(&"a"), Some(&1));
144     assert_eq!(bimap.insert_no_overwrite(2, "b"), Ok(()));
145     assert_eq!(bimap.get_right(&2), Some(&"b"));
146     assert_eq!(bimap.get_left(&"b"), Some(&2));
147     assert!(bimap.contains_left(&1));
148     assert!(bimap.contains_right(&"b"));
149 }
150
151 #[test]
152 /// 上書き挿入のテスト
153 /// 左側、右側、両側の上書きのケースを確認する
154 fn overwrite_insert() {
155     let mut bimap = BiHashMap::new();
156     assert_eq!(bimap.insert(1, "a"), OverwriteResult::NoOverwrite);
157     print_hash_map(&bimap, "insert (1, 'a')");
158     assert_eq!(bimap.insert(1, "b"), OverwriteResult::OverwriteRight("a"));
159     print_hash_map(&bimap, "insert (1, 'b')");
160     assert_eq!(bimap.get_right(&1), Some(&"b"));
161     assert_eq!(bimap.insert(2, "b"), OverwriteResult::OverwriteLeft(1));
162     print_hash_map(&bimap, "insert (2, 'b')");
163     assert_eq!(bimap.get_left(&"b"), Some(&2));
164     assert_eq!(bimap.get_right(&2), Some(&"b"));
165     assert_eq!(bimap.insert_no_overwrite(3, "c"), Ok(()));
166     print_hash_map(&bimap, "insert (3, 'c')");
167     assert_eq!(
168         bimap.insert(2, "b"),
169         OverwriteResult::OverwriteBoth((2, "b"), (2, "b"))
170     );
171     print_hash_map(&bimap, "insert (2, 'b') again");
172     assert_eq!(
173         bimap.insert(3, "b"),
174         OverwriteResult::OverwriteBoth((3, "c"), (2, "b"))
175     );
176     print_hash_map(&bimap, "insert (3, 'b')");
177     assert_eq!(bimap.get_right(&3), Some(&"b"));
178     assert_eq!(bimap.get_left(&"b"), Some(&3));
179     assert_eq!(bimap.left.len(), bimap.right.len());
180     assert_eq!(bimap.get_right(&2), None);
181     assert_eq!(bimap.insert_no_overwrite(3, "d"), Err(()));

```

```

182     print_hash_map(&bimap, "insert_no_overwright (3, 'd') [fail]");
183     assert_eq!(bimap.insert_no_overwrite(5, "e"), Ok(()));
184     print_hash_map(&bimap, "insert_no_overwright (5, 'e') [ok]");
185     assert_eq!(bimap.insert_no_overwrite(5, "d"), Err(()));
186     print_hash_map(&bimap, "insert_no_overwright (5, 'd') [fail]");
187 }
188
189 /// 左右の削除のテスト
190 #[test]
191 fn remove_test() {
192     let mut bimap = BiHashMap::new();
193     bimap.insert_no_check(1, "a");
194     bimap.insert_no_check(2, "b");
195     bimap.insert_no_check(3, "c");
196     print_hash_map(&bimap, "initial map");
197     assert_eq!(bimap.remove_left(&2), Some("b"));
198     print_hash_map(&bimap, "after remove_left(2)");
199     assert_eq!(bimap.get_left(&"b"), None);
200     assert_eq!(bimap.get_right(&2), None);
201     assert_eq!(bimap.remove_right(&"c"), Some(3));
202     print_hash_map(&bimap, "after remove_right('c')");
203     assert_eq!(bimap.get_left(&"c"), None);
204     assert_eq!(bimap.get_right(&3), None);
205     assert_eq!(bimap.remove_left(&4), None);
206     print_hash_map(&bimap, "after remove_left(4) [no op]");
207     assert_eq!(bimap.remove_right(&"d"), None);
208     print_hash_map(&bimap, "after remove_right('d') [no op]");
209 }
210
211 use std::fmt::Debug;
212 fn print_hash_map<R, L>(bimap: &BiHashMap<R, L>, mes: &str)
213 where
214     R: Debug + Hash + Eq + Clone,
215     L: Debug + Hash + Eq + Clone,
216 {
217     println!("=== {} ===", mes);
218     println!("Left to Right:");
219     for (l, r) in &bimap.left {
220         println!("  {:?} => {:?}", l, r);
221     }
222     println!("Right to Left:");
223     for (r, l) in &bimap.right {
224         println!("  {:?} => {:?}", r, l);
225     }
226 }
227 }

```